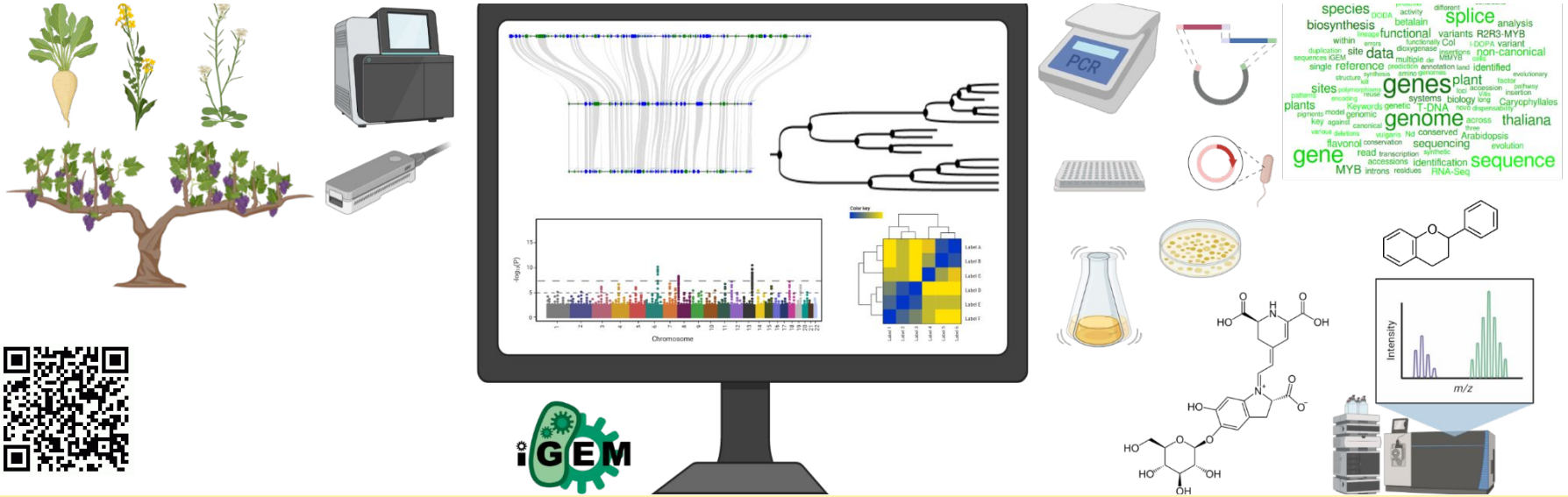
Technische Universität Braunschweig



Python - Application examples

Prof. Dr. Boas Pucker (Plant Biotechnology and Bioinformatics)

Availability of slides

- All materials are freely available (CC BY) - after the lectures:
  - StudIP: ‘Python for Life Scientists’
  - GitHub: https://github.com/bpucker/teaching

- Questions: Feel free to ask at any time

- Feedback, comments, or questions: b.pucker[a]tu-bs.de

Reverse complement of nucleotide sequence

What happens here?

```
1  def revcomp( seq ):
2
3      seq = seq.lower()
4
5      #key:value (=dictionary)
6      complement = { 'a':'t', 't':'a', 'c':'g', 'g':'c' }
7
8      new_seq = []
9
10     for nt in seq:
11         new_seq.append( complement[ nt ] )
12
13     #list[::-1] inverts list (last element becomes first)
14     new_seq = "".join( new_seq[::-1] )
15
16     return new_seq
```

Sequence of bases e.g. ATGACATGA

Converts input to lower case: atgacatga

Get complement for each base

Inverts list (=reverse)

Exercises - Part 6a

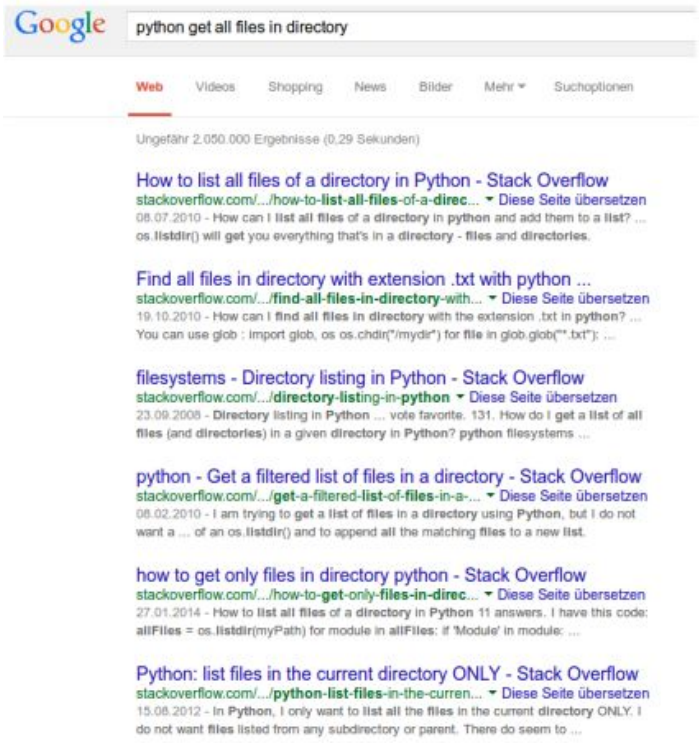
- 6.1) Write a function to get the reverse complement (upper case letters) of a DNA sequence given in upper case letters!
- 6.2) Write a function to convert a DNA sequence into a RNA sequence!
- 6.3) Write a function to translate a DNA sequence into amino acids (first frame only)!
- 6.4) Write a function to translate DNA sequences in all 6 frames into peptide sequences! The longest peptide sequence per DNA sequence should be returned!
- 6.5) Write a function to grep a sequence from a FASTA file based on the name of this sequence!

How to approach a challenge in bioinformatics?

- Identify the problem that needs to be solved
- Split the problem into smallest possible parts
- Solutions for small parts of the problem might be available already
- Precise description of the problem is required for online search
- Best hit often leads to stackoverflow:
  - Problem is described by the asking person
  - Multiple solutions are suggested by the community
  - Community votes to identify the best solution
  - Green marking highlights the answer that solved the problem

Example

- Problem: get paths of all files in a certain directory
- Search expression: ‘python get all files in directory’



Best hit on stackoverflow

533 user like this answer

This answers solved the question

14 Answers

active oldest votes

533

You can use `glob`:

```
import glob, os
os.chdir("/mydir")
for file in glob.glob("*.txt"):
    print(file)
```

or simply `os.listdir`:

```
import os
for file in os.listdir("/mydir"):
    if file.endswith(".txt"):
        print(file)
```

or if you want to traverse directory, use `os.walk`:

```
import os
for root, dirs, files in os.walk("/mydir"):
    for file in files:
        if file.endswith(".txt"):
            print(os.path.join(root, file))
```

share improve this answer

edited Apr 22 at 17:44 Tarantula 4,794 • 3 • 22 • 39

answered Oct 19 '10 at 1:12 ghostdog74 102k • 17 • 116 • 188

1 Using solution #2, How would you create a file or list with that info? – Merlin Oct 19 '10 at 3:48

35 @ghostdog74: In my opinion it would more appropriate to write `for file in f` than for `for files in f` since what is in the variable is a single filename. Even better would be to change the `f` to `files` and then the for loops could become `for file in files`. – martineau Oct 26 '10 at 14:18

18 @computermacgyver: No, `file` is not a reserved word, just the name of a predefined function, so it's quite possible to use it as a variable name in your own code. Although it's true that generally one should avoid collisions like that, `file` is a special case because there's hardly ever any need to to use it, so it is often consider an exception to the guideline. If you don't want to do that, PEP8 recommends appending a single underscore to such names, i.e. `file_`, which you'd have to agree is still quite readable. – martineau Oct 14 '12 at 19:04

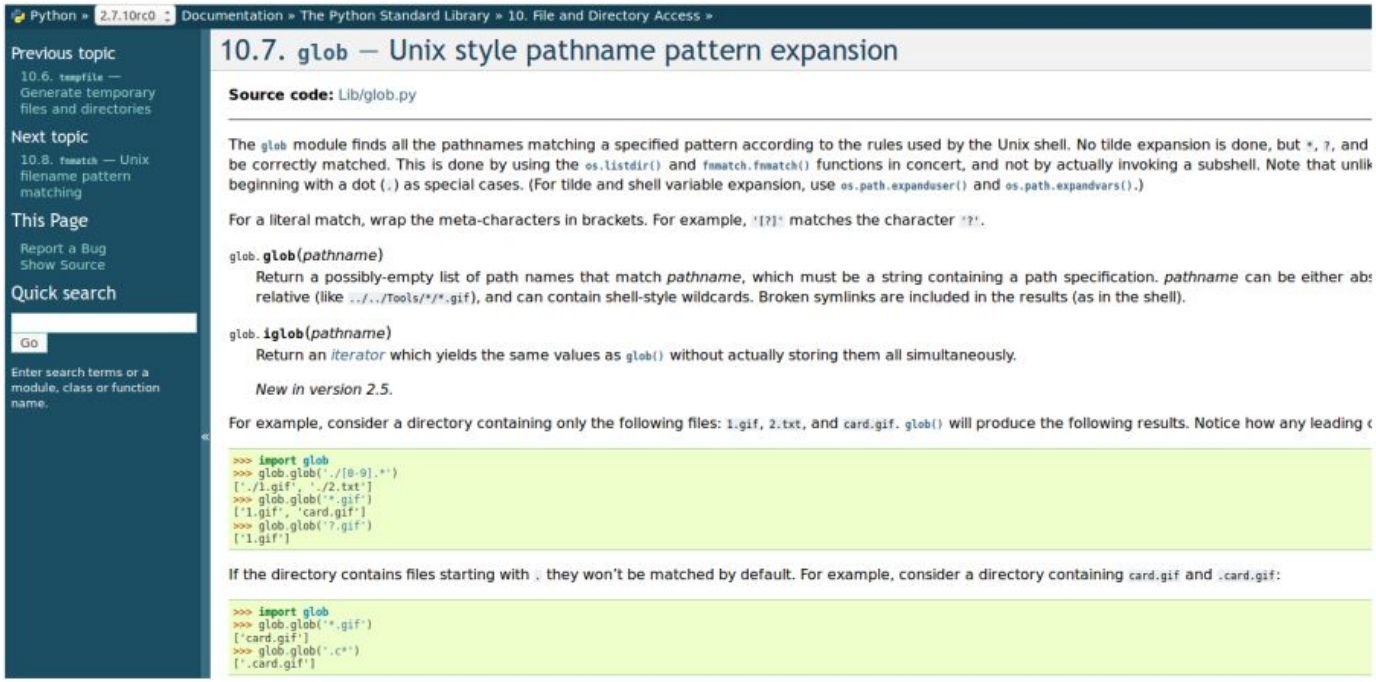
2 Thanks, martineau, you're absolutely right. I jumped too quickly to conclusions. – computermacgyver Oct 15 '12 at 19:53

1 Really cool answer, you could replace r,d,f by r,_,f to avoid unused variable declaration. – AsTeR Mar 8 '13 at 20:16

Three different ways to solve this problem are described

Official Python documentation

- Systematic documentation of all functions in a module with all possible arguments
- Sometimes examples are given



Python » 2.7.10rc0 Documentation » The Python Standard Library » 10. File and Directory Access »

Previous topic
10.6. tempfile — Generate temporary files and directories

Next topic
10.8. fnmatch — Unix filename pattern matching

This Page
Report a Bug
Show Source

Quick search
Go
Enter search terms or a module, class or function name.

10.7. glob — Unix style pathname pattern expansion

Source code: Lib/glob.py

The glob module finds all the pathnames matching a specified pattern according to the rules used by the Unix shell. No tilde expansion is done, but *, ?, and be correctly matched. This is done by using the os.listdir() and fnmatch.fnmatch() functions in concert, and not by actually invoking a subshell. Note that unlik beginning with a dot (.) as special cases. (For tilde and shell variable expansion, use os.path.expanduser() and os.path.expandvars().)

For a literal match, wrap the meta-characters in brackets. For example, '[?]' matches the character '?'.

glob.**glob**(*pathname*)
Return a possibly-empty list of path names that match *pathname*, which must be a string containing a path specification. *pathname* can be either abs relative (like ../../Tools/*/*.gif), and can contain shell-style wildcards. Broken symlinks are included in the results (as in the shell).

glob.**iglob**(*pathname*)
Return an *iterator* which yields the same values as glob() without actually storing them all simultaneously.

New in version 2.5.

For example, consider a directory containing only the following files: 1.gif, 2.txt, and card.gif. glob() will produce the following results. Notice how any leading c

```
>>> import glob
>>> glob.glob('./[0-9].*')
['./1.gif', './2.txt']
>>> glob.glob('*.gif')
['1.gif', 'card.gif']
>>> glob.glob('?.gif')
['1.gif']
```

If the directory contains files starting with . they won't be matched by default. For example, consider a directory containing card.gif and .card.gif:

```
>>> import glob
>>> glob.glob('*.gif')
['card.gif']
>>> glob.glob('.c*')
['.card.gif']
```

How to run BLAST?

- Running at the NCBI website does not give you full control over all parameters

- Python can be used to run a local search:

```
blastn \
-query <query_file> \
-subject <subject_file> \
-out <output_file> \
-outfmt 6 \
-evalue 0.01 \
-word_size 4
```

1.	qseqid	query (e.g., gene) sequence id
2.	sseqid	subject (e.g., reference genome) sequence id
3.	pident	percentage of identical matches
4.	length	alignment length
5.	mismatch	number of mismatches
6.	gapopen	number of gap openings
7.	qstart	start of alignment in query
8.	qend	end of alignment in query
9.	sstart	start of alignment in subject
10.	send	end of alignment in subject
11.	evalue	expect value
12.	bitscore	bit score

https://www.metagenomics.wiki/tools/blast/blastn-output-format-6

How to execute processes via shell?

- Running shell commands through the subprocess module:
  p = subprocess.Popen(arg='ls -lh', shell=True)
  p.communicate()

- Can be used to run everything via Python

- Python waits until the command is completed

- Example:

```
import subprocess
p = subprocess.Popen( arg="mkdir test && cd test && ls -lh", shell=True )
p.communicate()
```

How to process BLAST results?

```python
def load_BLAST_results( input_file ):
    """! @brief load all BLAST results from file """

    data = []
    with open( input_file, "r" ) as f:
        line = f.readline()
        while line:
            parts = line.strip().split('\t')
            data.append( {  'query': parts[0],
                            'subject': parts[1],
                            'query_start': int( parts[6] ),
                            'query_end': int( parts[7] ),
                            'score': float( parts[-1] )
                        } )
            line = f.readline()
    return data
```

```
AT1G01010       NdCChr1.g1.t1   100.00  429     0       0       1       429     1       429     0.0      895
AT1G01010       NdCChr4.g18734.t1       32.84   469     247     15      1       428     2       443     4e-56    194
AT1G01010       NdCChr1.g127.t1 34.23   336     157     10      1       330     1       278     1e-41    152
AT1G01010       NdCChr1.g128.t1 32.38   349     157     14      1       331     1       288     4e-39    146
AT1G01010       NdCChr4.g18730.t1       39.33   178     95      5       1       175     2       169     1e-32    126
AT1G01010       NdCChr3.g12773.t1       39.63   164     89      2       1       162     1       156     1e-28    115
AT1G01010       NdCChr4.g22969.t1       40.74   162     79      5       5       159     11      162     3e-28    117
AT1G01010       NdCChr4.g18733.t1       40.00   165     90      5       1       162     2       160     4e-28    115
AT1G01010       NdCChr3.g17122.t1       42.31   156     74      6       5       153     15      161     4e-27    112
```

How to organize a Python script

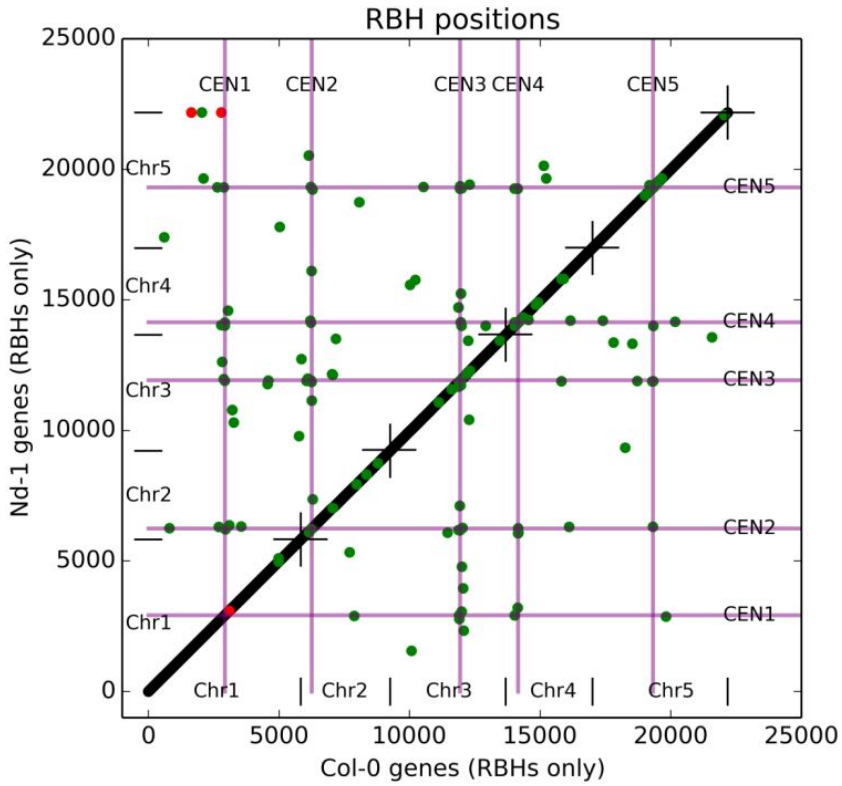
- Make a script recognize that it needs to run with Python:
  #!/usr/bin/env python3

- Other information to include:
  - Author
  - Version
  - Usage
  - Imports

```
1  ### Boas Pucker ###
2  ### bpucker@cebitec.uni-bielefeld.de ###
3  ### v0.2 ###
4
5  __usage__ = """
6                  python construct_RNA_seq_coverage_file.py\n
7                  --in <BAM_FILE>
8                  --out <OUTPUT_FILE>
9
10                 --bam_is_sorted <PREVENTS_EXTRA_SORTING_OF_BAM_FILE>
11
12                 feature requests and bug reports: bpucker@cebitec.uni-bielefeld.de
13                 """
14
15 __cite__ = """ Pucker & Brockington, 2018: https://doi.org/10.1186/s12864-018-5360-z """
16
17
18 import os, sys
19
20 # --- end of imports --- #
21
22 def main( arguments ):
```

matplotlib

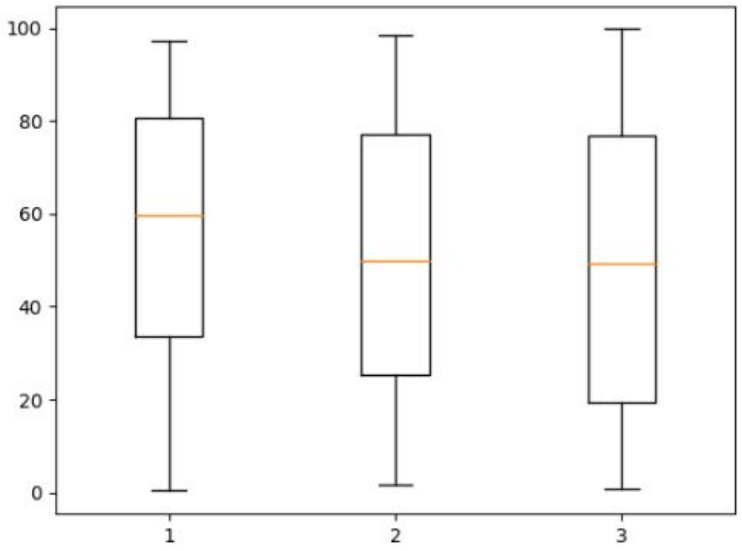
- Importing matplotlib:
  import matplotlib.pyplot as plt
- Visualization of complex data
- Automatic generation of plots
- Unlimited customization options



Pucker et al., 2016: https://doi.org/10.1371/journal.pone.0164321

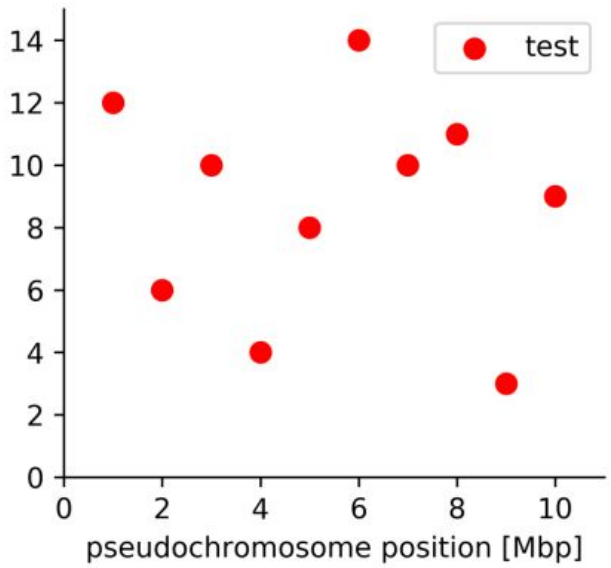
Box plot

```
1  import matplotlib.pyplot as plt
2  import numpy as np
3
4  d1 = np.random.rand(50) * 100   #generate random numbers
5  d2 = np.random.rand(50) * 100
6  d3 = np.random.rand(50) * 100
7
8  data = [d1, d2, d3] # multiple box plots on one figure
9
10 plt.figure()
11 plt.boxplot(data)
12 plt.show()
```



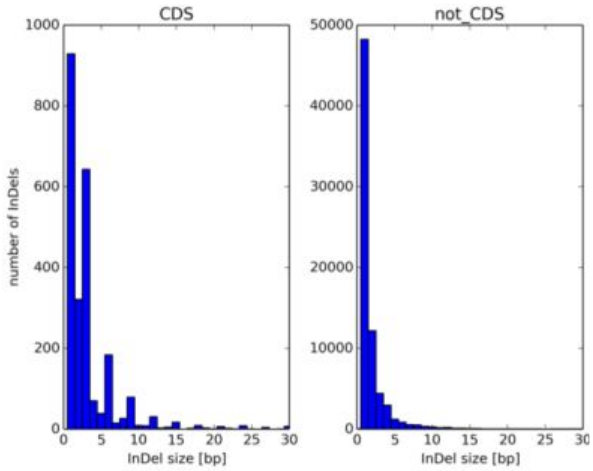
Scatter plot

```
1   import matplotlib.pyplot as plt
2
3   fig, ax = plt.subplots( figsize=( 10, 4 ) )   #defining size of plot
4
5   x_values = [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ]
6   y_values = [ 12, 6, 10, 4, 8, 15, 10, 11, 3, 9 ]
7
8   ax.scatter( x_values, y_values, color="red", s=10, marker="o", label="test" )
9   #setting color, marker size, marker shape and label of this group
10
11  ax.legend( numpoints=1 )
12  #each group is represented by only one marker in the legend (default=3)
13
14  ax.set_xlim( 0, 11 )    #set range of x-axis
15  ax.set_ylim( 0, 15 )    #set range of y-axis
16
17  ax.set_xlabel( "pseudochromosome position [Mbp]" )
18
19  ax.spines["top"].set_visible(False)      #remove lines and ticks
20  ax.spines["right"].set_visible(False)    #remove lines and ticks
21
22  plt.subplots_adjust(left=0.05, right=0.99, top=0.97, bottom=0.12)
23  #adjust size of plot within figure
24
25  plt.show()
26  fig.savefig( "my_plot.png", dpi=600 )   #write figure into output file
27  plt.close( "all" )   #destroy created figures (cleaning up)
```



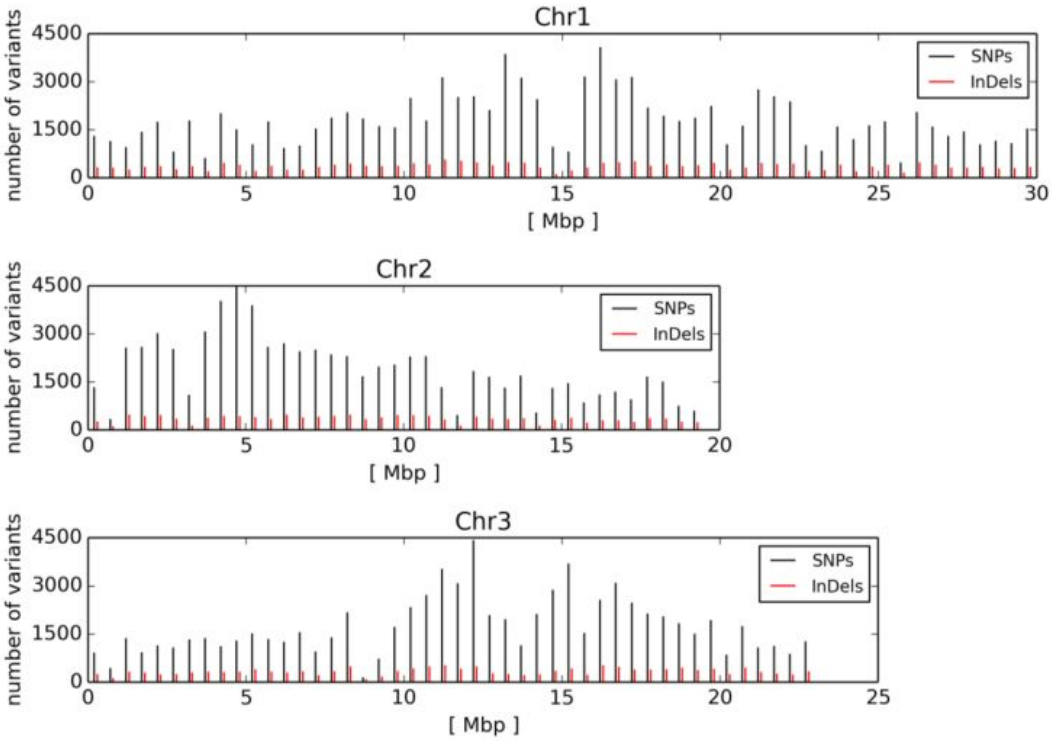
Histogram

```
1  import matplotlib.pyplot as plt
2
3  # --- end of imports --- #
4
5  gene_space = [  3, 3, 6, 6, 9, 9, 12, 3, 3, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
6                  11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
7                  12, 15, 18, 21, 24, 27, 30 ]
8  intergenic = [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1, 2, 3, 4, 5, 6, 7, 8, 9,
9                  1, 2, 3, 4, 5, 6, 7, 1, 2, 3, 4, 1, 2, 1 ]
10
11
12 fig, ( ax1, ax2 ) = plt.subplots( 1, 2, sharey=False)
13 counts, bins, patches = ax1.hist( gene_space, bins=max( gene_space ), align="left" )
14 ax1.set_title( "CDS" )
15 ax1.set_xlim( 0, 30 )
16 ax1.set_xlabel( "InDel size [bp]" )
17 ax1.set_ylabel( "number of InDels" )
18
19 counts, bins, patches = ax2.hist( intergenic, bins=max( intergenic ), align="left" )
20 ax2.set_title( "not_CDS" )
21 ax2.set_xlim( 0, 30 )
22 ax2.set_xlabel( "InDel size [bp]" )
23 plt.subplots_adjust( wspace=0.3 )   #increase space between figures
24
25 plt.show()
26 fig.savefig( prefix + "InDel_size_distribution.png", dpi=300 )
27 plt.close('all')
```



Pucker et al., 2016: https://doi.org/10.1371/journal.pone.0164321

Barplot figure



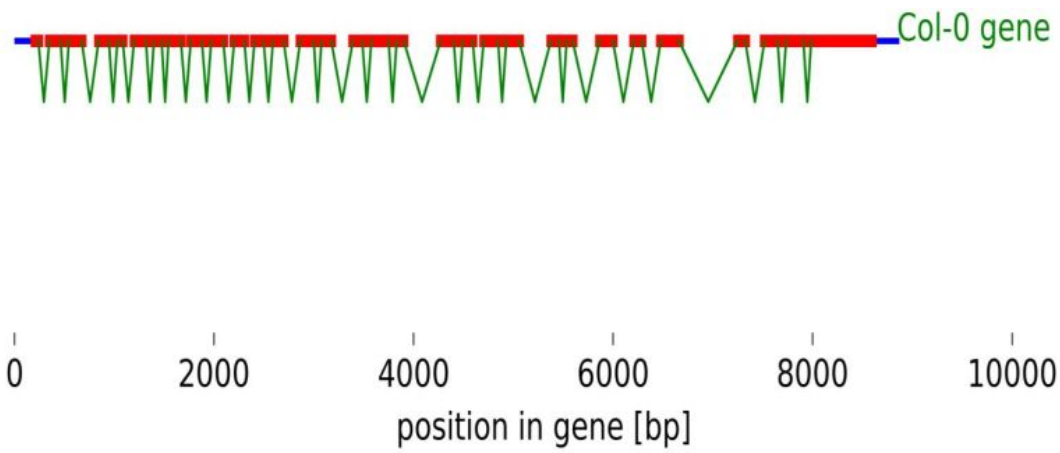
barplots.py generates barplots at specific positions by drawing a normal line

(script is available in course repository)

Pucker et al., 2016: https://doi.org/10.1371/journal.pone.0164321

Gene structure plot

- gene_structure_plot.py generates visualizations of gene/transcript structures based on GFF annotations



Exercises - Part6b

- 6.6) Construct a figure to illustrate the order and orientation of genes in the gum gene cluster of *Xanthomonas campestris* pv. campestris!

- 6.7) Save this figure in different file formats (png, jpg, pdf, svg)!

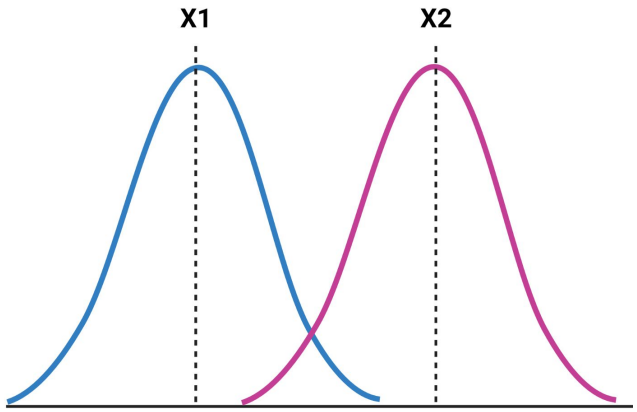
Statistics

- Compare observed sample against the expected distribution
- Check if two samples are derived from same distribution

- H0 = samples were taken from same distribution

- H0 can only be rejected or kept due to insufficient evidence against it

- H0 can NEVER be confirmed



Shapiro-Wilk test

- Testing data set for normal distribution
- Important for decision about potential tests

```
from scipy import stats
x = [1, 2, 3, 3, 3, 2, 1]
stats.shapiro(x)
```

Correlation

- Pearson correlation coefficient is suitable for data following a normal distribution
- Spearman correlation coefficient is better if data distribution is unknown

```
from scipy import stats
x = [1,2,3,4,5]
y = [2,4,6,8,10]
r,p = stats.pearsonr(x,y)
r,p = stats.spearmanr(x,y)
```

t-test

- Samples need to show normal distribution
- Comparison of one sample against a reference value
- Comparison of two samples:
  - Paired samples (ttest_rel)
  - Unpaired samples (ttest_ind)

```
from scipy import stats
x = [1,2,3,4,5]
y = [4,6,8,10,11]
t,p = stats.ttest_ind(x,y) #independent samples
t,p = stats.ttest_rel(x,y) #paired samples
```

W-test

- Wilcoxon (W) test compares two paired samples
- Normal distribution is not required

```
from scipy import stats
x = [1,2,3,4,5]
y = [4,6,8,10,11]
w,p = stats.wilcoxon(x,y)
```

U-test

- Comparison of unpaired samples
- Normal distribution is not required

```
from scipy import stats
x = [1,2,3,4,5]
y = [4,6,8,10,11]
w,p = stats.mannwhitneyu(x,y)
```

Chi square test

- Comparison of an observation against an expectation
- Comparisons of two observations

```
from scipy import stats
obs = [1,2,3,4,5]
exp = [4,6,8,10,11]
x, p = stats.chisquare(obs,exp)
```

Exercises - Part6c

- 6.8) Construct a suitable visualization!
- 6.9) Analyze distribution and trends!
- 6.10) Apply statistical test to investigate difference!

HTML

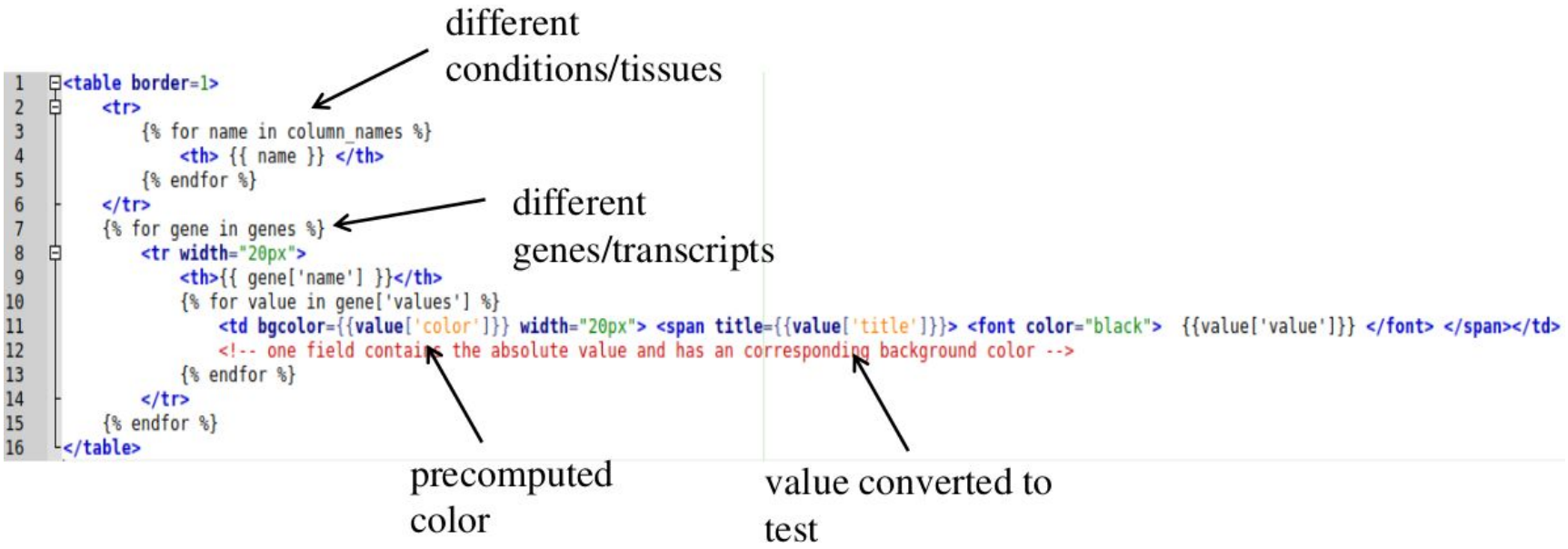
- Construction of a HTML-based heatmap

gene ID	0H	4H	Salt	Heat	Inflorescence	leaf	Root	seedlings_HiK	seedlings_HiK2	Leaf_SSC
(p	0.02	0.0	0.0	0.0	0.0	0.0	4.33	1.44	1.2	0.0
(c	0.0	0.0	0.0	0.0	0.0	0.0	1.17	0.11	0.0	0.0
(c	0.0	0.0	0.0	0.0	0.0	0.0	3.0	0.11	0.4	0.0
p	1.76	29.23	14.65	6.11	203.75	265.2	0.83	199.33	199.7	54.4
o	0.0	0.0	0.0	0.01	1.38	0.0	0.0	0.0	0.1	0.0
(	0.0	0.03	0.01	0.0	0.25	0.0	0.0	0.0	0.1	0.02
y	0.01	0.01	0.0	0.06	13.88	0.0	0.0	0.11	0.1	0.01
u	6.25	58.89	41.42	15.61	69.5	5.4	3.83	157.22	163.1	22.49
h	0.86	5.04	0.09	0.24	33.5	0.8	12.67	8.78	8.5	1.07
y	10.74	43.0	4.78	6.45	29.0	1.2	7.17	62.67	67.0	25.48
e	19.29	14.22	6.14	5.21	4.88	1.6	11.5	28.11	24.2	13.68
w	35.33	45.16	52.79	70.1	17.5	74.2	6.67	21.89	22.6	65.95
q	6.03	24.4	2.69	2.38	80.0	35.6	19.83	55.22	59.9	11.16
k	5.74	15.9	1.35	4.56	57.5	3.0	1.17	9.44	7.5	8.47
s	0.27	1.74	0.07	0.25	11.38	0.2	0.0	0.78	0.4	0.11
i	0.04	0.0	0.0	0.0	36.0	0.0	0.0	0.89	1.1	0.0
	0.0	0.0	0.0	0.0	18.0	0.0	0.0	0.67	0.7	0.0
	9.84	14.92	17.27	7.68	16.13	27.0	26.0	26.89	23.7	11.58
e	17.99	9.41	7.94	8.45	11.38	21.2	13.83	19.22	19.7	14.81

HTML template

- construct_heatmap.py reads values from text file and prepares data structures to fill this template

- HTML document can be converted to PDF




```
1  <table border=1>
2      <tr>
3          {% for name in column_names %}
4              <th> {{ name }} </th>
5          {% endfor %}
6      </tr>
7      {% for gene in genes %}
8          <tr width="20px">
9              <th>{{ gene['name'] }}</th>
10             {% for value in gene['values'] %}
11                 <td bgcolor={{value['color']}} width="20px"> <span title={{value['title']}}> <font color="black">  {{value['value']}} </font> </span></td>
12                 <!-- one field contains the absolute value and has an corresponding background color -->
13             {% endfor %}
14         </tr>
15     {% endfor %}
16 </table>
```

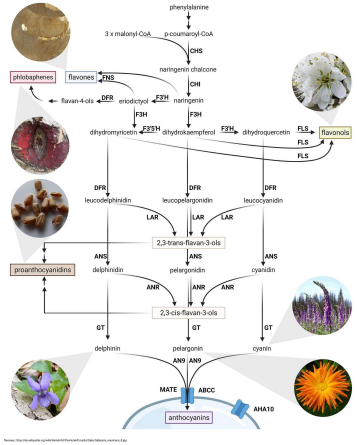
Exercises - Part6d

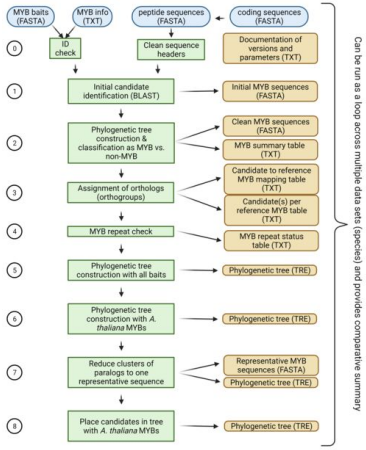
- 6.11) Read data table and construct heatmap for the gene expression in HTML!
- 6.12) Add mouse-over effect to display functional gene annotation!

Looking for more opportunities to apply Python?

- Plant Biotechnology and Bioinformatics group is working on:
  - Plant genomics
  - Plant transcriptomics (RNA-seq)
  - Specialized plant metabolites
  - Synthetic biology
  - Big data comparative studies
  - Tool development

- Details: https://www.tu-braunschweig.de/en/ifp/pbb





https://doi.org/10.1186/s12864-022-08452-5

Time for questions!

References (biological background of examples)

- Nd-1 genome assembly (Pucker *et al*., 2016)
  - https://doi.org/10.1371/journal.pone.0164321
- Non-canonical splice sites (Pucker *et al*., 2017)
  - https://doi.org/10.1186/s13104-017-2985-y
- *Croton tiglium* transcriptome assembly (Haak *et al*., 2018)
  - https://doi.org/10.3389/fmolb.2018.00062
- Genome-wide non-canonical splice sites in plants (Pucker & Brockington, 2018)
  - https://doi.org/10.1186/s12864-018-5360-z
- Chromosome-level Nd-1 genome assembly (Pucker *et al*., 2019)
  - https://doi.org/10.1371/journal.pone.0216233
- NAVIP (Baasner *et al*., 2019)
  - https://doi.org/10.1101/596718